An Energy-Efficient Network-on-Chip Design using Reinforcement Learning

Hao Zheng
George Washington University
haozheng@gwu.edu

Ahmed Louri
George Washington University
louri@gwu.edu

ABSTRACT

The design space for energy-efficient Network-on-Chips (NoCs) has expanded significantly comprising a number of techniques. The simultaneous application of these techniques to yield maximum energy efficiency requires the monitoring of a large number of system parameters which often results in substantial engineering efforts and complicated control policies. This motivates us to explore the use of reinforcement learning (RL) approach that automatically learns an optimal control policy to improve NoC energy efficiency. First, we deploy power-gating (PG) and dynamic voltage and frequency scaling (DVFS) to simultaneously reduce both static and dynamic power. Second, we use RL to automatically explore the dynamic interactions among PG, DVFS, and system parameters, learn the critical system parameters contained in the router and cache, and eventually evolve optimal per-router control policies that significantly improve energy efficiency. Moreover, we introduce an artificial neural network (ANN) to efficiently implement the large state-action table required by RL. Simulation results using PARSEC benchmark show that the proposed RL approach improves power consumption by 26%, while improving system performance by 7%, as compared to a combined PG and DVFS design without RL. Additionally, the ANN design yields 67% area reduction, as compared to a conventional RL implementation.

ACM Reference Format:
Hao Zheng and Ahmed Louri. 2019. An Energy-Efficient Network-on-Chip Design using Reinforcement Learning. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3317768

1 INTRODUCTION

Network-on-Chips (NoCs) have emerged as the standard interconnect fabric to connect cores, last level caches, and memory modules on the chip. Current many-core chips consist of tens to hundreds of cores, and future projections call for thousands of cores. However, NoCs consume a substantial portion (approximately 10 % to 36 %) of the total chip's power [1–3]. The power problem will only become more of a challenge with the continuous scaling of transistor feature size, and as more power is consumed in communication than in computing. Therefore, a number of studies have been proposed to increase power savings for future NoCs design.

To achieve maximum power savings, it is imperative to combine the benefits of various techniques in an integrated manner. Power-gating (PG) is an effective technique to reduce static power [4, 5]. Dynamic voltage and frequency scaling (DVFS) is an effective technique to reduce dynamic power [6, 7]. A combined design, PG and DVFS, offers static and dynamic power reduction benefits. However, the dynamic interactions between these techniques and the network could have negative effects. An overestimated DVFS decision incurs wasted dynamic power. On the other hand, an underestimated DVFS decision could result in network congestion, which in turn reduces the idle cycles between flits. Recall that the PG decision is made on these idle cycles, thus it negatively affects the static power reduction. This calls for a proactive control policy to avoid such negative effects. Moreover, exploring various application behavior may improve decision accuracy and power savings, but it complicates the control policy. Applications have varying impacts on the behavior of the NoC, cache and miss status holding registers (MSHRs) [8, 9]. At the NoC level, the applications result in different network utilization and message information [10, 11]. For example, the network messages, such as load/store and response/request, indicate the data criticality. At the cache level, L1 instruction and data cache activities are correlated to the computation intensity, while L2 cache and MSHR activities provide information about NoC traffic. Collecting this information could increase decision accuracy and power reduction benefits, but it complicates the control policy and requires enormous engineering efforts.

Traditional control algorithms, and more recently, supervised learning have been proposed to optimize NoC design [12–15]. A Proportional-Integral-Derivative (PID) controller monitors the output variances, then computes the logic by calculating proportional, integral, and derivative values. However, the empirically tuned parameters could fail to resist variations in applications and uncertainties of NoC behavior. Similarly, supervised learning requires human understanding to create labeled training examples prior to the training phase, thus requiring human engineering.

In this paper, we propose a reinforcement learning approach which automatically learns an optimal policy to map runtime system parameters for optimal decision making. First, we propose a combined design—PG and DVFS—to simultaneously reduce static and dynamic power. PG disconnects the router from the power supply when the router is idle. DVFS dynamically tunes the V/F levels and thus provides varying power levels. Second, we propose an RL-based control policy for the combined design. A per-router based RL agent learns a set of system parameters related to the NoC and cache, and eventually evolves an optimal per-router control policy. By automatically and optimally exploring these system parameters

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317768

and actions, the trained control policy maximizes energy efficiency. Moreover, RL requires the use of a state-action table to record states, actions, and the rewards resulting from taking actions. When a large number of system parameters are used, the state-action table increases significantly, which makes its hardware cost prohibitive. In this paper, we propose an artificial neural network (ANN) to reduce the hardware costs of implementing the RL on a per-router basis. Our simulation results show that our proposed approach can effectively reduce overall power consumption by 26%, and improve performance by 7%. The ANN design achieves 67% area reduction, as compared to a conventional RL implementation.

2 PROPOSED DESIGN

2.1 Architecture Overview

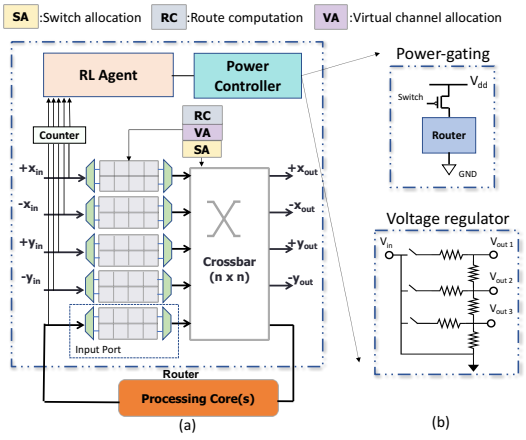

Figure 1: (a) Micro-architecture of the proposed router, and (b) PG technique and voltage regulator.

In the proposed architecture, we assume an 8×8 mesh topology with deterministic XY routing. The processing core with a private L1 cache and a shared L2 cache is attached to the router. When a cache miss occurs, the local private L1 first checks whether it hosts the data. If not, a request message is initialized and then sent to shared L2 caches via the NoC. Since wormhole routing is assumed, a packet is segmented into several flits in NoC for efficient router resource utilization.

Figure 1(a) shows a 4-stage router comprised of virtual channels (VCs) for storing arriving flits, Routing Computation (RC) for calculating flit route, Virtual Channel Allocation (VA) for assigning virtual channel and flow control, and Switch Allocation (SA) for allocating an input port on internal crossbar. Our proposed router design has two additional units called the RL agent and the power controller.

The RL agent maps system parameters to an optimal action about which V/F level to take, and updates the state-action table based on reward function. We implement a set of counters to collect system parameters, and record them in the state-action table. These parameters include the number of flits received at each port, the number of response flits, the number of request flits, and the number of cache misses. By collecting these parameters, the RL agent could have a better understanding of application behavior, thus yielding a more accurate control policy. Moreover, the RL agent monitors PG performance by counting the number of powered off cycles, which will be an indication of the PG and DVFS interaction.

The power controller consists of PG and DVFS controller. The PG disconnects the router from the power supply using a header transistor, shown in Figure 1(b), as a switch to turn on/off the router. When the power controller detects a number of consecutive idle cycles, it cuts off the router from the V_{dd} to save static power. In this paper, we assume 4 cycles as the detection time [16]. The controller resumes the router's full activity when detecting an incoming flit (any incoming flit on any port will wake up the router). The voltage regulator, shown in Figure 1(b), selects voltage levels for each router. In this paper, we assume voltage levels from 0.6V to 1V, and frequency from 1GHz to 2GHz. Upon receiving a decision from the RL agent, the DVFS controller selects the appropriate V/F for saving router's dynamic power every 10K cycles. We set the voltage regulator transition latency as 100 ns [9]. Since V/F levels may be different among routers, the dual-clock First-input, First-out (FIFO) synchronizer is used for router-to-router communication.

2.2 RL-Based Control Policy

2.2.1 *Reinforcement Learning Basics.*

Reinforcement learning (RL) [17] is a machine learning approach, in which the *agent* acts as a learner and decision maker by interacting with the *environment*. Figure 2 shows the dynamic interaction between the RL agent and the environment. (1) The agent selects an action a_t from a set of actions, $\mathcal{A} = \{1, ..., K\}$, at time step t. (2) The selected action influences the environment by affecting the internal *state* s_t and *rewards* r_t. (3) This eventually results in a new state and reward, s_{t+1} and r_{t+1}, at the next time step $t + 1$.

In RL, the goal of the agent is to interact with the environment by selecting actions in a way that maximizes the long-term total rewards $\mathcal{R}$, which is the cumulative sum of all future rewards, as Equation 1. The future rewards $(r_{t+1}, r_{t+2}, ...)$ are discounted by a factor of γ $(0 \leq \gamma \leq 1)$ called the *discount factor*. As γ approaches to 0, the agent becomes near-sighted and only considering current rewards.

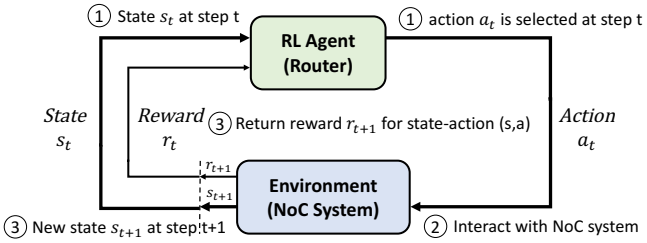

Figure 2: The agent-environment interaction in RL.

$$\mathcal{R} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \quad (1)$$

All RL algorithms involve estimating the action-value function to select an optimal action with the highest reward $\mathcal{R}$. In other words, it is essential to estimate the immediate rewards r for an agent to be in a given state. Since the rewards depend on what

actions the agent will take, value functions are defined with respect to particular ways of acting, called policies π. The value of taking action a in state s is denoted Q(s,a).

Tabular Q-learning is one of the RL algorithms, which finds optimal Q-value function. A Q-value table is initialized with random values for all possible (s,a) pairs. At each time step, the Q-learning algorithm chooses actions based on current Q, such that, over many time steps, all actions are taken in all states. In each time step, the action-value table entry Q(s,a) is updated using Equation 2 based on action a, reward r, and new state s'.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma maxQ(s',a) - Q(s,a)] \quad (2)$$

where α is learning rate and γ is discount factor and $maxQ(s',a)$ is the maximum Q value over all possible actions in state s'.

2.2.2 Problem Formulation.

Per-router RL Agent: The RL agent uses the monitored system parameters and decides which voltage and frequency action to take every 10K cycles.

Actions: The action space consists of different V/F levels that routers can choose from. For clarity, this paper has three actions, $\mathcal{A} = \{a_0, a_1, a_2\}$, which are 2GHz/1V, 1.5GHz/0.8V, and 1GHz/0.6V, respectively. The number of actions can be obviously much larger.

Category	State Attributes	Description
Cache Related Metrics	1. L1D cache miss	the number of L1 data cache misses
	2. L1I cache miss	the number of L1 instruction cache misses
	3. L2 cache miss	the number of L2 cache misses
Network Related Metrics	4. +X buffer utilization	the buffer utilization of +X input port
	5. –X buffer utilization	the buffer utilization of -X input port
	6. +Y buffer utilization	the buffer utilization of +Y input port
	7. -Y buffer utilization	the buffer utilization of –Y input port
	8. Local port buffer utilization	the buffer utilization of local port
	9. Router throughput	the number of flits received per epoch
Message Information	10. Response flits	the number of response flits received
	11. Request flits	the number of request flits received
PG and DVFS Interaction	12. PG efficiency	the efficiency of power-gating ($T_{power\text{-}off}/T_{epoch}$)

Figure 3: The state attributes used in RL.

State Space: A state s is a vector of system attributes. In this paper, it consists of cache and network related metrics as shown in Figure 3.

- *Cache Related Metrics:* State attributes 1-3 are used to represent the cache activities of a local core. The activities of L1 cache are correlated to the computation intensity. As L2 caches are shared and distributed across the chip, they communicate through the NoC. These communications are a good indication of the overall NoC global behavior.
- *Network Related Metrics:* State attributes 4-9 indicate the number of received flits at each port. These are used by the RL agent to monitor different routing directions. In NoCs, flits are often categorized into two classes called response and request (state attributes 10-11). Response flits often have priority over the request flits due to the message dependence. State attribute 12 is the PG efficiency of a router, indicating the efficiency of the PG for different DVFS decisions.

Reward Function: RL agent uses reward function to evaluate how beneficial it is to take a specific *action* for a given *state*. Typically, choosing an action with higher reward function tends to result in better system performance (*e.g.* power savings). Therefore, the goal of RL agent is to maximize the long-term reward, which in our case implies maximizing overall power savings. Thus, we design the reward function for a router as:

$$Reward = P_{static} + P_{dynamic} - P_{pg\ overhead} \quad (3)$$

P_{static} and $P_{dynamic}$ represent the static and dynamic power savings of the given router, respectively, while $P_{pg\ overhead}$ is the power overhead resulting from powering on the router.

To avoid undesired performance loss due to DVFS, we apply a negative reward when we observe that the average read cache miss latency exceeds the threshold of performance loss. To calculate this average read cache miss latency, we record the time difference between the issued and completed time of each MSHR entry (read cache miss). The reward is updated either according to Equation (3) when the latency is under the threshold, or assigned a negative reward equals to -1 to prevent the performance loss. A negative reward that is smaller than -1 can be applied to further improve performance at the expense of power savings.

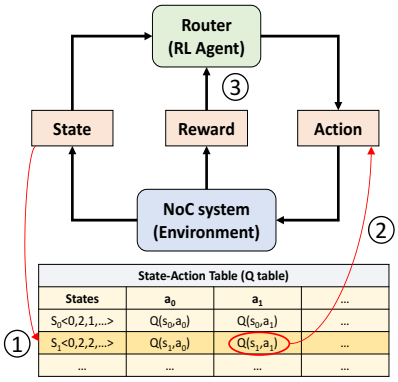

Figure 4: The procedure of updating stat-action table following Q-learning rules.

State-Action Table: RL agent selects actions according to the Q-values of a given state. Q-values for any state-action pair are recorded independently in a per-router based state-action table. An example is shown in Figure 4. The RL agent observes states and records them as a vector <0,2,2,...> in a given time t, where each element of the vector represents a specific state. Since some state attributes are continuous numbers, they could lead to infinite state space and infeasible Q-learning converging time. Therefore, those continuous numbers are discretized into finite bins. For example, the RL agent monitors that the router was powered off for 5000 cycles in the past 10K-cycle epoch, which means its PG efficiency is 0.5 (power off time/epoch time). In this paper, each state attribute has a discrete set of 5 bins, {0,1,2,3,4}. Therefore, the PG efficiency of 0.5 is denoted as 2.

Walkthrough Example: The entire process of RL is shown in Figure 4: (1) The observed state vector <0,2,2,...> is used to query the table where S_1 matches the state <0,2,2,...>. The action a_1 has the maximum Q-value in the matched entry, thus a_1 is selected. We note that ε-greedy policy is used to ensure continual design space exploration, in which a random action may be selected with a small probability of ε. (2) Upon taking this action, the system

transits to state s' in the next time step t'. (3) By following the Q-learning update rule as Equation 2, a Q-value is learned from the past experience. The total reward incorporates the immediate reward r that results from the action a_1 and the discounted future reward $\gamma maxQ(s', a')$.

Learning parameters: In RL, the learning rate α, the discount rate γ, and ϵ can be tuned. The learning rate α is set to 0.1. For the γ, a larger number would lead to a policy more focused on future rewards. In this paper, we define γ of 0.95 as a value with optimal performance. When ϵ approaches zero, the RL stops exploration of unknown actions and selects actions based on current policy. We empirically find that ϵ of 0.1 provides optimal power savings.

2.3 ANN for Reducing RL Implementation Overheads

In RL, each state vector consists of a number of states, as discussed in Section 2.2. When the RL agent observes any new state-action pair, it creates a new entry in the state-action table to record its actions and associated Q-values. Even though we have discretized the parameters into finite bins for a smaller table size, our simulation results show that the state-action table still requires over 10% router area to store all state-action pairs. To address this problem, we replace the state-action table with an offline-trained ANN for less hardware costs. The ANN calculates the state-action table instead of storing the entire state-action table in the router, thus eliminating the storage space for state-action pairs.

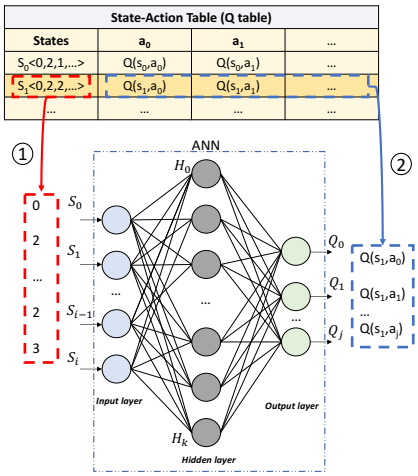

Figure 5: The ANN calculates the Q-values instead of using a state-action table.

ANN Architecture: The proposed ANN consists of 3 layers: input layer, hidden layer and output layer. Each layer consists of multiple neurons. The Sigmoid and Relu functions are used for the hidden and output layers, respectively. The sizes of input and output layers depend on the designs of the state and action space. More detailed discussion on the hidden layer size will be discussed in Section 3.5.

Training Details: In the offline training, the ANN takes each state-action entry as a training sample. The state vector is used as the input of ANN, and Q values are used as the desired output values. Note that input values are normalized in range of 0 to 1 for the nonlinearity of sigmoid function. We use the mean square error function to calculate the error between ANN's output and desired values, and then use the mini-batch gradient descent approach to back propagate this error to the hidden layer to tune their weights. We note that the batch size is set as 100 in this work. For an offline approach, we use 0.001 as the learning rate, because the accuracy is more important than the speed.

Walkthrough Example: Figure 5 shows a walkthrough example of using ANN. (1) S_1<0,2,2,...> is monitored by an RL agent, which is used as the input values of the ANN. (2) The network calculates the input values layer by layer and then delivers three values to the output layer, $Q(s_1, a_0)$, $Q(s_1, a_1)$, and $Q(s_1, a_2)$. The action with the highest Q-value will be selected.

3 EVALUATION AND RESULTS

3.1 Experimental Setup

Table 1: Key Simulation Parameters

# of cores	64 out-of-order CPUs, ALPHA, 2GHz
Router	4-stage pipeline
Cache block size	64 Bytes
Virtual channel	2 VCs/VN
Input buffer size	1-flit for control and 3-flit for data
Protocol	MESI
Memory latency	128 cycles
Topology	8×8 Mesh

We evaluate the proposed architecture under full system simulation with the combined use of architecture-level and circuit-level simulators. The cycle-accurate gem5 simulator [18] enhanced with GARNET [19] is used for detailed timing simulation of the memory and on-chip network. We use DSENT to evaluate power consumption. We also use Synopsys design compiler with 45nm library to evaluate the area overheads. Table 1 shows the specific parameters used in the simulation. We analyze our framework with PARSEC [20] benchmark suite.

The proposed RL-based policy is initialized as follows: the Q-values are initialized as 0; RL parameters are selected as discussed in Section 2.2.2.

The simulation framework contains:

(1) Baseline: each NoC router has PG and DVFS logic. The V/F level responds to the router's throughput. The thresholds are set as 0.1 flits/cycle and 0.05 flits/cycle.
(2) PID: A conventional PID controller is implemented with NoC router. The controller takes average waiting time of flits in the input buffer as a reference. The PID, a feedback control theory, decides the future DVFS decision through its integral and derivative. The integral provides the prediction based on previous history, while the derivative predicts its future trends.
(3) QL: each NoC router has Q-learning algorithm with an ANN for the proposed design.

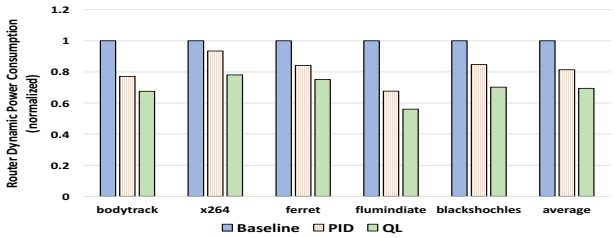

Figure 6: Dynamic power consumption, normalized to the baseline.

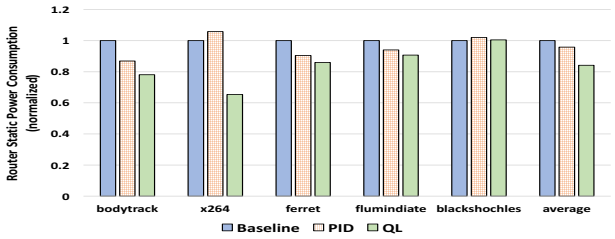

Figure 7: Static power consumption comparison, normalized to the baseline.

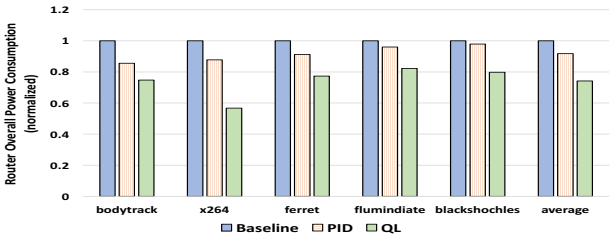

Figure 8: Overall power consumption comparison, normalized to the baseline.

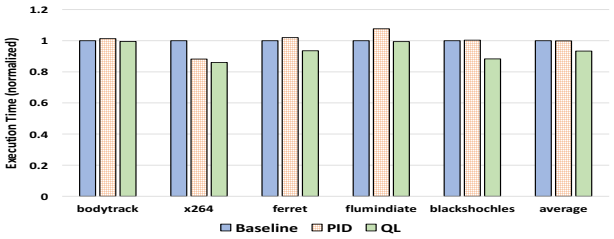

Figure 9: Application execution time comparison, normalized to the baseline.

3.2 Power Consumption Analysis

3.2.1 Dynamic Power Consumption Analysis. Figure 6 shows the dynamic power consumption analysis. It can be seen that the proposed design achieves an average of 30% dynamic power savings over the baseline. Even though a PID controller has an average of 17% dynamic power reduction, the proposed design can improve upon the PID design by 13%.

3.2.2 Static Power Consumption Analysis. Figure 7 shows the static power consumption analysis. It can be seen that the proposed solution reduces the average static power consumption by 16% as compared to the baseline.

3.2.3 Overall Power Consumption Analysis. Figure 8 shows the overall power consumption results. It can be seen that the proposed solution has an average of 26% power reduction compared to the baseline. Moreover, the proposed design can improve overall power consumption upon the PID design by 17%.

3.3 Performance Analysis

Figure 9 shows the performance analysis. The performance is obtained by measuring the application execution time of PARSEC benchmarks. It can be seen that the proposed solution has an average of 7% performance improvement over the baseline, with the best case achieving 14% performance improvement.

3.4 Overhead Analysis

3.4.1 Timing Overhead for ANN calculation. We define the timing overhead as the time that ANN calculates the Q-values. The timing overhead depends on the amount of hardware resources in the RL agent. In this paper, we assume one adder and one multiplier for the ANN which consists of 12 neurons at the input layer, 20 neurons at the hidden layer, and 3 neurons at the output layer. With such design, the ANN calculation incurs 299 ns latency. This latency can be overlapped by a large epoch equals to 10K cycles.

3.4.2 Area Overhead for Implementing RL. For the area overhead, the neural network requires an ALU (e.g. Multiplier, Sigmoid function) and storage. We evaluate the hardware cost through Synopsis Design Vision using 45 nm technology. The proposed ANN uses SRAM to store the weights that required for each layer (Input, Hidden and Output layers). Recall that the ANN has 12 input neurons, 20 hidden neurons and 3 output neurons, which implies that the ANN has 300 weights in this work. We use 20 bits to store a single weight. The ANN requires 300×20 bits in total which is smaller than 1KB. This consumes 0.55 mw with an area overhead of 4620 um^2. In addition, simulation results show that ALU and counters consume 2733 um^2.

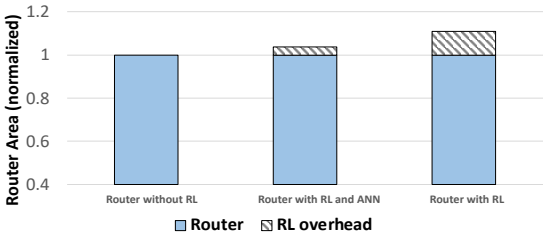

Figure 10: Router area comparison, normalized to the baseline without RL design.

Figure 10 shows the normalized area analysis. The baseline is the area of a non-RL router design. It can be seen that the ANN design can reduce the area overhead by 67%, as compared to the RL design using state-action table. The ANN only requires an area increase of 3.7%, as compared to the baseline.

3.5 Impacts of ANN size on RL performance

The number of neurons used in the hidden layer of ANN can affect the accuracy of calculating Q(s,a), which could affect power savings. In order to obtain the optimal network size, we study the the ANN accuracy using a different number of neurons for the hidden layer. Figure 11 shows that 5, 10, 15 and 20 neurons for the hidden layer can have 74%, 82%, 83% and 97% accuracy, as compared to a state-action table, respectively. It shows that the accuracy of this network can approach 1 when the hidden layer size equals to 20.

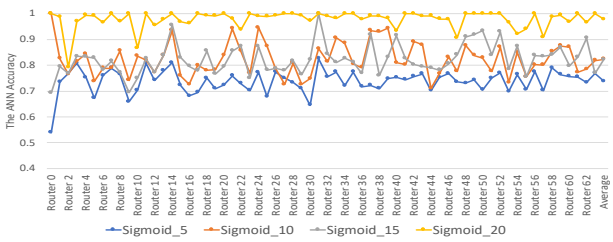

Figure 11: Accuracy analysis of the ANN using different sizes of the hidden layer.

4 RELATED WORKS

PG: Powerpunch [16] improves PG efficiency by sending an early wake-up signal to the powered-off router while the NI is processing the packet. In doing so, the full wake-up latency is hidden. NoRD [4] provides a bypass ring network to bypass powered-off routers, while avoiding waking up the powered-off routers. In [23], authors use a simple switch to bypass flits while the router is powered off.

DVFS: In [21], it shows that DVFS can be effectively applied on on-chip voltage scaling by using on-chip voltage regulators. Shang et al. [6] use DVFS to change the voltage and frequency levels of individual links according to historical link and input buffer utilizations. In [7], Chen et al. propose a throughput-driven and a latency-based PI controller, both with dynamic reference points. In [22], authors use cache-coherence communication properties to predict the NoC traffic and set voltage and frequency.

RL in NoCs: Yin et al. [24] proposed a self-learned NoC arbitration policy that can effectively reduce packet latency. Wang et al. [25] used the RL to balance the trade-offs among NoC performance, energy efficiency and reliability.

5 CONCLUSIONS

In this paper, we proposed an RL-based energy-efficient NoC design. The approach combines PG and DVFS, in order to reduce both static and dynamic power consumption. The RL algorithm automatically observes the effects of the combined techniques and various network parameters, updates a control policy that dynamically changes V/F levels, and selects the actions which yield maximum power savings and maintain system performance. The proposed RL approach improves power consumption by 26% while improving system performance by 7%, as compared to a combined PG and DVFS design without RL. We also proposed an offline-trained ANN to reduce the prohibitive hardware costs using RL. The proposed ANN design reduces 67% area overhead of implementing RL in NoC routers.

6 ACKNOWLEDGE

This research was partially supported by NSF grants CCF-1547035, CCF-1547036 and CCF-1702980. We thank the anonymous reviewers for their excellent feedback.

REFERENCES

[1] TG Mattson and et al. The 48-core scc processor: The programmer's view. In *Intl. Symp. on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–11, 2010.
[2] Y. Hoskote, S. Vangala, A. Singha, N. Borkar, and S. Borkar. A 5-ghz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5):51–61.
[3] V. Ganesh, S. Jack, G.Nathan, G. Saturnino, B. Vladyslav, L. Jose, S. Steven, and T. Bedford. Conservation cores: reducing the energy of mature computations. *ACM SIGARCH Computer Architecture News*, 38(1):205–218, 2010.
[4] L.Chen and T.M. Pinkston. NoRD: Node-router decoupling for effective power-gating of on-chip routers. In *Intl. Symp. on Microarchitecture (MICRO)*, Feb. 2012.
[5] R. Das, S. Narayanasamy, SK Satpathy, and RG Dreslinski. Catnap: Energy proportional multiple network-on-chip. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 320–331. ACM, 2013.
[6] L. Shang, L. Peh, and N. Jha. Power-efficient interconnection networks: Dynamic voltage scaling with links. *IEEE Computer Architecture Letters*, 1(1):6–6, 2002.
[7] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, U. Ogras, and R. Ayoub. Dynamic voltage and frequency scaling for shared resources in multicore processor designs. In *Design Automation Conference (DAC)*, 2013.
[8] N. Barrow-Williams, C. Fensch, and S. Moore. A communication characterisation of splash-2 and parsec. 2009.
[9] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras. In-network monitoring and control policy for dvfs of cmp networks-on-chip and last level caches. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(4):47, 2013.
[10] R. Hesse, J. Nicholls, and NE Jerger. Fine-grained bandwidth adaptivity in networks-on-chip using bidirectional channels. In *Intl. Symp. on Networks on Chip (NoCS)*, pages 132–141. IEEE, 2012.
[11] Y. Yao and Z. Lu. Dvfs for nocs in cmps: A thread voting approach. In *Intl. Symp. on High Performance Computer Architecture (HPCA)*, pages 309–320, 2016.
[12] Y. Wang, M. Martonosi, and L. Peh. A supervised learning approach for routing optimizations in wireless sensor networks. In *International workshop on Multi-hop ad hoc networks: from theory to reality*, pages 79–86. ACM, 2006.
[13] J. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou. Up by their bootstraps: Online learning in artificial neural networks for cmp uncore power management. In *Intl. Symp. on High Performance Computer Architecture (HPCA)*, pages 308–319. IEEE, 2014.
[14] D. Juan, S. Garg, J. Park, and D. Marculescu. Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling. In *Intl. Conf. on Hardware/Software Codesign and System Synthesis*, page 8, 2013.
[15] E. Kakoulli, V. Soteriou, and T. Theocharides. Intelligent hotspot prediction for network-on-chip-based multicore systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(3):418–431, 2012.
[16] L. Chen, D. Zhu, M. Pedram, and TM Pinkston. Power punch: Towards non-blocking power-gating of noc routers. In *Intl. Symp. on High Performance Computer Architecture (HPCA)*, pages 378–389. IEEE, 2015.
[17] R. Sutton, A. Barto, et al. *Reinforcement learning: An introduction*. MIT, 1998.
[18] N. Binkert and et al. The gem5 simulator. In *ACM SIGARCH Computer Architecture News*, May 2011.
[19] N. Agarwal, T. Krishna, L. Peh, and N. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Intl. Symp. on Performance Analysis of Systems and Software (ISPASS)*, pages 33–42, 2009.
[20] C. Bienia and K. Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *5th Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.
[21] W. Kim, MS. Gupta, G. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *Intl. Symp. on High Performance Computer Architecture (HPCA)*, pages 123–134, 2008.
[22] R. Hesse and N. E. Jerger. Improving dvfs in nocs with coherence prediction. In *Intl. Symp. on Networks-on-Chip*, page 24. ACM, 2015.
[23] H. Zheng and A. Louri. Ez-pass: An energy & performance-efficient power-gating router architecture for scalable nocs. *IEEE Computer Architecture Letters*, 17(1):88–91, 2018.
[24] J. Yin, Y. Eckert, S. Che, M. Oskin, and G. Loh. Toward more efficient noc arbitration: A deep reinforcement learning approach. In *Proceedings of the 1st International Workshop on AI-assisted Design for Architecture (AIDArc)*, 2018.
[25] K. Wang, A. Louri, A. Karanth, and R. Bunescu. High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learning. In *Proceedings of the 22nd Design, Automation & Test in Europe (DATE) conference*, 2018.